

# OSGi

## Conception d'applications modulaires en Java

**Jérôme Molière**

Préface de Peter Kriens



# OSGi

## Conception d'applications modulaires en Java

OSGi spécifie un ensemble de services afin de concevoir des applications modulaires tant dans le domaine de l'embarqué que dans celui des applications d'entreprise classiques et serveurs.

### Modularité et services : une autre façon de développer en Java

Le développeur Java qui souhaite s'affranchir des limitations des ClassLoader en environnement J2EE, prévenir les phases d'intégration longues et risquées, et satisfaire les contraintes de disponibilité de son application, trouvera des réponses à ses préoccupations dans la façon dont OSGi spécifie des services modulaires. L'architecture OSGi s'est déjà imposée dans de nombreux projets, tels que l'environnement de développement Eclipse Equinox, les serveurs d'applications GlassFish 3, JOnAS 5 ou JBoss 5...

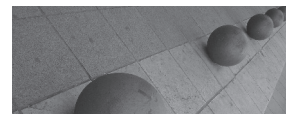
Sans entrer dans des détails d'implémentation spécifique, ce livre éclaire les principes de modularité et d'orientation par services qui sont au cœur d'OSGi. Il guide ainsi le développeur Java vers une méthodologie de conception et une façon nouvelle de penser ses développements. L'auteur préconise des bonnes pratiques en matière d'architecture mais aussi de tests et d'intégration continue, tout en donnant des conseils sur l'outillage à adopter, pour envisager sereinement les contraintes du travail en équipe et bénéficier d'avantages tels que la mise à jour à chaud des modules d'une application.

### Au sommaire

**OSGi pour une conception modulaire** • Problèmes liés au paquetage des applications J2EE • Regrouper des bibliothèques au sein d'un conteneur OSGi • Vers les architectures orientées services • Dépasser les limitations des ClassLoaders Java • **Couplage et injection de dépendances** • **Les bundles OSGi** • Les spécifications OSGi • OSGi comme socle architectural de serveur : JOnAS5 • OSGi pour les applications embarquées • **Mettre en place son premier bundle OSGi** • Installation de bndtools • Méta-informations nécessaires • Bundles et bibliothèques Java utilisés • Chargement de classes • Encapsulation de la bibliothèque Commons Lang • **Services OSGi par défaut** • Publier son premier service OSGi • Gestion du dynamisme avec le tracking de services • **Environnement et outils** • Conteneurs OSGi • Apache Felix • Knopflerfish • La console OSGi d'Eclipse : Equinox • **Nécessité d'un référentiel de bundles : OBR** • Architecture et dynamique d'appels avec OBR • **Les plug-ins de développement Eclipse pour OSGi** • bnd et bndtools • IPOJO • PDE • **Le moteur d'injection de dépendances : la spécification Declarative Services** • Le SCR (Service Component Runtime) • Code source et descripteurs de déploiement • **Services OSGi pour le développeur** • Gestion des traces avec le LogService • Communication par messages avec l'EventAdmin • Exposer ses ressources sur le Web avec l'HttpService • Service de configuration : le ConfigAdmin • **Modularité et tests unitaires** • Du point de vue du chef de projet, du programmeur, de l'architecte • **Modularité et tests en environnement OSGi** • Gestion des versions de composants en environnement OSGi • **Design patterns OSGi** • Séparation entre API et implémentations • Le design pattern Factory • Le design pattern du tableau blanc ou Whiteboard • Invocation dynamique de code • Exemple de situation : sérialisation/désérialisation de données avec CastorXML • **Application exemple** • **Annexe** • Mettre en place un bundle OSGi avec PDE.

### À qui s'adresse cet ouvrage ?

- À tous les développeurs Java EE qui souhaitent mettre en place OSGi ;
- Aux architectes logiciels et chefs de projets qui veulent intégrer OSGi dans leurs projets pour bénéficier de la programmation modulaire.



### Jérôme Molière

Ingénieur certifié Sun et JBoss, **Jérôme Molière** a plus de 15 ans d'expérience en développement Java. Il intervient dans diverses conférences et événements nationaux et internationaux (OSGi Con, RMLL, Solutions Linux), est architecte indépendant depuis 2004 au sein de Mentor/J ([jerome@javaxpert.com](mailto:jerome@javaxpert.com)) et intervient auprès de grands comptes sur des problématiques Java/J2EE : coaching d'équipe et formation, audit, tuning, optimisation et intégration continue. Il est l'auteur du *Cahier du programmeur J2EE* aux éditions Eyrolles.

Code éditeur : G13328  
ISBN : 978-2-212-13328-8

Conception : Nord Compo

# OSGi

**Conception d'applications  
modulaires en Java**

## Du même auteur

J. MOLIÈRE. – **Conception et déploiement Java/J2EE**  
N°11194, octobre 2003, 192 pages.

## Dans la même collection

S. JABER. – **Programmation GWT 2. Développer des applications RIA et Ajax avec le Google Web Toolkit.**  
N°12569, 2010, 484 pages.

J. PAULI, G. PLESSIS, C. PIERRE DE GEYER. – **Audit et optimisation LAMP.**  
N°12800, 2012, 300 pages environ.

R. RIMELÉ. – **HTML 5. Une référence pour le développeur web.**  
N°12982, 2011, 604 pages.

F. DAOUST, D. HAZAËL-MASSIEUX. – **Relever le défi du Web mobile. Bonnes pratiques de conception et de développement.**  
N°12828, 2011, 300 pages.

J. CHABLE, D. GUIGNARD, E. ROBLES, N. SOREL. – **Programmation Android.**  
N°13303, 2<sup>e</sup> édition, 2012, 520 pages environ.

T. SARLANDIE, J.-M. LACOSTE. – **Programmation IOS 5 pour iPhone et iPad.**  
N°12799, 2<sup>e</sup> édition, 2012, 350 pages environ.

E. SARRION. – **jQuery Mobile. La bibliothèque JavaScript pour le Web mobile.**  
N°13388, 2012, 610 pages.

J. STARK. – **Applications iPhone avec HTML, CSS et JavaScript. Conversion en natifs avec PhoneGap.**  
N°12745, 2010, 190 pages.

E. DASPET et C. PIERRE DE GEYER. – **PHP 5 avancé.**  
N°13435, 6<sup>e</sup> édition, 2012, 900 pages environ.

D. SEGUY, P. GAMACHE. – **Sécurité PHP 5 et MySQL.**  
N°13339, 3<sup>e</sup> édition, 2011, 277 pages.

P. BORGHINO, O. DASINI, A. GADAL. – **Audit et optimisation MySQL 5.**  
N°12634, 2010, 282 pages

C. PORTENEUVE. – **Bien développer pour le Web 2.0. Bonnes pratiques Ajax.**  
N°12391, 2<sup>e</sup> édition, 2008, 674 pages.

J.-M. DEFRANCE. – **Ajax, jQuery et PHP.**  
N°13271, 3<sup>e</sup> édition, 2011, 482 pages.

## Autres ouvrages

V. MESSENGER ROTA. – **Gestion de projet agile. Avec Scrum, Lean, eXtreme Programming...**  
N°12750, 3<sup>e</sup> édition, 2010, 272 pages.

A. BOUCHER. – **Ergonomie web illustrée. 60 sites à la loupe.**  
N°12695, 2010, 302 pages (Design & Interface).

A. BOUCHER. – **Ergonomie web. Pour des sites web efficaces.**  
N°13215, 3<sup>e</sup> édition, 2011, 356 pages (Accès libre).

I. CANIVET. – **Bien rédiger pour le Web. Stratégie de contenu pour améliorer son référencement naturel.**  
N°12883, 2<sup>e</sup> édition, 2011, 552 pages (Accès libre).

M.-V. BLOND, O. MARCELLIN, M. ZERBIB. – **Lisibilité des sites web. Des choix typographiques au design d'information.**  
N°12426, 2009, 326 pages (Accès libre).

E. SLOÏM. – **Mémento Sites web. Les bonnes pratiques.**  
N°12802, 3<sup>e</sup> édition, 2010, 18 pages.

O. ANDRIEU. – **Réussir son référencement web. Édition 2012.**  
N°13396, 4<sup>e</sup> édition, 2011, 700 pages.

G. SWINNEN. – **Apprendre à programmer avec Python 3.**  
N°12708, 2<sup>e</sup> édition, 2010 (Collection Noire).

E. SARRION – **jQuery & jQuery UI.**  
N°12892, 2011, 520 pages (Collection Noire).

## Collection « A Book Apart »

J. KEITH, préface de J. ZELDMAN. – **HTML 5 pour les Web Designers.**  
N°12861, 2010, 90 pages.

D. CEDERHOLM. – **CSS 3 pour les Web designers.**  
N°12987, 2011, 132 pages.

E. KISSANE. – **Stratégie de contenu web.**  
N°13279, 2011, 96 pages.

E. MARCOTTE. – **Responsive Web Design.**  
N°13331, 2011, 160 pages.

A. WALTER. – **Design émotionnel.**  
N°13398, 2011, 110 pages.



# OSGi

**Conception d'applications  
modulaires en Java**

**Jérôme Molière**

Préface de Peter Kriens

**EYROLLES**

A horizontal line with a small grey circle in the center, positioned below the publisher's name.

ÉDITIONS EYROLLES  
61, bd Saint-Germain  
75240 Paris Cedex 05  
[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

*Remerciements de l'éditeur à Anne Bougnoux pour sa relecture,  
ainsi qu'à Peter Kriens, Vincent Beretti et Christophe Peigné.*

*Préface de Peter Kriens, traduite de l'anglais par l'éditeur.*

# Préface

---

Pour nous, qui venons du Nord, la France méridionale a bien des attraits. La douceur du climat, la bonne chère et le bon vin, les jolies femmes et les interactions sociales riches... Bref, tout un *art de vivre* ! Mon poste de Directeur technique chez OSGi m'a conduit à parcourir toute la planète, sans me laisser le loisir de séjourner trop longtemps en Suède. Malgré l'envie de températures plus douces, quitter la Suède n'a pas été un choix facile parce qu'elle est étrangement attachante, pour un pays si froid. Pourtant, en 2003, nous avons franchi le pas et emménagé dans un endroit magnifique, Beaulieu, dans la campagne environnant Montpellier.

Parmi les choses qui m'ont manqué, les échanges avec mes collègues d'Ericsson et les déjeuners rituels dont je garde un très bon souvenir. En effet, toute attrayante que soit la côte méditerranéenne, on n'y trouve que très peu de développeurs OSGi... Aussi fus-je stupéfait d'apprendre que l'un des conférenciers acceptés en 2009 à la conférence OSGi DevCon à Jazoon habitait à moins de 12 kilomètres de chez moi ! Je lui ai écrit pour proposer un déjeuner – quel rêve en effet que de tenir une discussion approfondie sur les ClassLoaders Java tout en jouissant des fruits de « La Campagne » ! C'est ainsi que j'ai rencontré Jérôme, qui m'a non seulement fait découvrir les meilleures frites du monde (à Lansargues), mais est devenu un compagnon régulier du Bon Coin.

J'ai ainsi pu mesurer les efforts titanesques que demande l'écriture d'un livre. Au cours des dix dernières années, j'ai été sollicité de nombreuses fois pour rédiger des ouvrages sur la création logicielle, mais le travail est infernal, peu rémunéré et tout à fait sous-estimé. Je tire mon chapeau à tous les auteurs qui sont capables d'aller au bout d'une telle entreprise. En effet, il suffit au lecteur d'une minute sans effort pour parcourir un texte qui a nécessité des heures muettes de dur labeur. Le processus de rédaction a confirmé l'idée que je me faisais de ce travail ; je suis maintenant admiratif qu'il soit terminé.

Enfin un ouvrage en français sur OSGi ! Il était temps, après la parution d'un ouvrage en allemand, un en chinois et plusieurs en anglais, quand on sait que les racines d'OSGi se situent en France et que cette norme y est assez populaire. J'y vois une sorte de reconnaissance !

Lorsque nous avons démarré OSGi, nous sommes partis du JES (*Java Embedded Server*) de Sun comme structure de base pour nos spécifications. L'auteur de ce produit, Anselm Baird-Smith, est un Français de Grenoble. S'il a quitté le projet quelques mois après, il n'en demeure pas moins pour moi l'auteur original du cœur de la norme. Après dix années passées à travailler sur son modèle, je reste béat d'admiration pour la simplicité et l'élégance du modèle OSGi original, et plus encore à présent que je réalise combien les purs programmeurs Java ont de mal à appréhender la modularité et comme ils tendent à créer un monstre dès qu'ils en ont la possibilité.

Certains lecteurs seront surpris qu'on puisse voir en OSGi une solution simple et élégante car des développeurs se plaisent à le dénigrer sur le Web en critiquant sa complexité. Mais quand on analyse ces attaques verbales, on se rend compte qu'ils reprochent à l'outil de ne pas faire... ce pour quoi il n'est pas fait ! C'est que la modularité ne s'utilise pas comme une bibliothèque qu'il n'y aurait qu'à ajouter dans son classpath... La modularité est une qualité de la structure de base de votre code. OSGi, lui, n'est qu'un messenger donnant l'alerte, lorsque votre code n'est pas modulaire.

Les mécontents d'OSGi ne se plaignent au fond que du fait que leurs hacks non modulaires échouent dans un environnement OSGi !

Malheureusement, la plupart des bibliothèques les plus utilisées ne sont pas modulaires dans leur conception. Il faudrait, pour jouir des avantages de la modularité, un remaniement en profondeur (*refactoring*) afin d'aménager la communication entre modules. C'est ce que propose OSGi avec son modèle par micro-services et ses dépendances par paquets : la solution la plus simple pour étendre Java de façon modulaire, sans toutefois perdre une sûreté de typage à laquelle les développeurs Java sont attachés.

Pour comprendre OSGi, il faut être prêt à changer de paradigme, à penser différemment. Or, un tel changement est toujours mystérieux : avant l'épiphanie, tout semble inutilement complexe et bruité. Mais une fois de l'autre côté, tout s'éclaire comme par magie et la cause des problèmes entrevus auparavant devient péniblement évidente. J'en ai fait l'expérience plusieurs fois dans ma vie, avec la programmation objet, la programmation fonctionnelle (avec Prolog !), l'inversion de contrôle... non sans peiner à chaque fois. Pour pimenter le tout, impossible de franchir la distance sur de simples explications, puisqu'on ne peut comprendre qu'à partir de ce qu'on sait déjà.

Voilà pourquoi ce livre est aussi important. Il offre une introduction pratique à OSGi avec de nombreux exemples, sans oublier d'en approfondir les aspects fondamentaux, à savoir les dépendances de paquets et les micro-services. Lorsque vous en lirez les explications, et en testerez les exemples, le chemin vous semblera ardu. Mais les leçons en valent la peine car elles vous donneront un précieux recul sur votre code, ainsi que sur la manière de l'optimiser et de le rendre plus modulaire.

# Table des matières

---

## **Avant-propos ..... 1**

### CHAPITRE 1

## **OSGi pour une conception modulaire..... 3**

OSGi, pour quelles applications ? .....	3
Cas d'une automobile .....	4
Cas d'un environnement de développement (IDE) .....	5
Cas d'un serveur d'applications .....	7
Problèmes liés au packaging des applications J2EE .....	8
Regrouper des bibliothèques au sein d'un conteneur OSGi .....	8
Vers les architectures orientées services .....	9
Dépasser les limitations dues aux ClassLoader Java .....	10
Tour d'horizon de la norme OSGi et de ses services .....	14
Couplage et injection de dépendances ? .....	15
Les bundles OSGi .....	16
Les spécifications OSGi en détail .....	16
Implémentations OSGi .....	17

### CHAPITRE 2

## **Exemples d'applications modulaires basées sur OSGi ..... 19**

OSGi comme socle architectural de serveur : JOnAS 5 .....	21
Le shell OSGi sous JOnAS .....	22
Un bref survol du code source de JOnAS .....	24
OSGi pour les applications embarquées : serve@home .....	29
Le projet domotique serve@home de Siemens .....	29
OSGi dans les automobiles .....	29

### CHAPITRE 3

## **Mettre en place son premier bundle OSGi ..... 31**

Installation de bndtools .....	32
Création du projet et développement du bundle .....	32

Création du projet .....	33
Méta-informations nécessaires .....	34
<b>Bundles et bibliothèques Java utilisés .....</b>	<b>39</b>
OSGi et chargement de classes .....	40
Encapsulation de la bibliothèque Commons Lang .....	40
<b>Services OSGi par défaut .....</b>	<b>43</b>
Publier son premier service OSGi .....	44
Gestion du dynamisme avec le tracking de services .....	50
S'appuyer sur les services standards de la plate-forme .....	51

## CHAPITRE 4

### **Travailler avec OSGi : environnement et outils ..... 55**

<b>Conteneurs OSGi .....</b>	<b>56</b>
Apache Felix .....	56
Le projet Knopflerfish .....	56
La console OSGi d'Eclipse : Equinox .....	57
Choisir sa console .....	58
<b>Nécessité d'un référentiel de bundles : OBR .....</b>	<b>58</b>
OSGi et bundles : limitations .....	59
Vers un entrepôt commun de bundles avec OBR .....	60
Premier contact avec OBR .....	60
Architecture et dynamique d'appels avec OBR .....	63
<b>Les plug-ins de développement Eclipse pour OSGi .....</b>	<b>64</b>
Simplifier la création de bundles avec bnd .....	64
Travailler avec bnd .....	65
Utiliser bndtools .....	66
iPOJO : une autre approche du développement OSGi .....	67
PDE : l'environnement standard de développement de plug-in d'Eclipse .....	67

## CHAPITRE 5

### **Le moteur d'injection de dépendances :**

### **la spécification Declarative Services..... 69**

<b>Introduction à la déclaration de services dans OSGi .....</b>	<b>70</b>
Le SCR (Service Component Runtime) .....	70
<b>Fonctionnalités proposées par le SCR .....</b>	<b>71</b>
Activation immédiate ou retardée ? .....	71
<b>Principes pour la mise en œuvre de SCR .....</b>	<b>73</b>
<b>Exemple complet .....</b>	<b>76</b>
Descriptif du produit .....	76
Entités en présence .....	76

Application en utilisation .....	77
L'exemple vu du côté technique .....	77
<i>Conception</i> .....	77
<i>Code source et descripteurs de déploiement</i> .....	78
Précisions sur le Declarative Services .....	85

## CHAPITRE 6

### **Services OSGi pour le développeur : LogService, EventAdmin, HttpService et ConfigAdmin..... 87**

Gestion des traces avec le LogService .....	88
Communication par messages avec l'EventAdmin .....	92
Présentation .....	92
Fonctionnalités et limitations .....	93
Principales entités mises en jeu .....	94
Exemple de mise en œuvre .....	95
Exposer ses ressources sur le Web avec l'HttpService .....	98
Service de configuration : le ConfigAdmin .....	100
Présentation .....	101
Conclusion .....	108

## CHAPITRE 7

### **Modularité et tests unitaires..... 109**

Retour sur la notion de modularité .....	110
Du point de vue du chef de projet .....	110
Du point de vue du programmeur .....	112
Du point de vue de l'architecte .....	115
Modularité et tests en environnement OSGi .....	117
Les commandes du shell .....	117
Tests en environnement OSGi .....	119
<i>Principe du test unitaire en environnement OSGi</i> .....	119
<i>Création de l'environnement d'exécution des tests unitaires</i> .....	120
Mise en pratique : création d'un test case pour un service OSGi simple .....	121
Considérations avancées .....	121
Gestion des versions de composants en environnement OSGi .....	125
Qu'est-ce qu'une version d'un composant et d'une application ? .....	125
La vision offerte par OSGi en matière de gestion de versions .....	126
Adopter une numérotation cohérente et en tirer parti .....	127
Gestion de la politique de numérotation des versions en pratique .....	128
Ce qu'il faut retenir .....	128

## CHAPITRE 8

**Design patterns OSGi et bonnes pratiques ..... 129**

Gabarits de conception .....	129
Séparation entre API et implémentations .....	129
Le design pattern Factory .....	130
Le design pattern du tableau blanc ou Whiteboard .....	130
<i>Comment fonctionne le WhiteBoard ?</i> .....	133
Invocation dynamique de code dans le contexte OSGi .....	137
<i>Exemple de situation : sérialisation/désérialisation de données</i> <i>avec Castor XML</i> .....	137
<i>Mise en place pratique</i> .....	140
Conseils et pièges à éviter .....	145
Shells OSGi et niveaux de démarrage .....	145
Gestion de la problématique des versions .....	146
Injection de dépendances .....	146
Maîtrise de la méta-information .....	147

## CHAPITRE 9

**Application exemple ..... 149**

L'application exemple .....	150
Vue d'ensemble de l'architecture .....	150
Notes relatives à la conception de l'application .....	152
<i>Création des PDF</i> .....	152
<i>Planification de tâches en Java</i> .....	153
Quelques images du produit .....	154
Notes relatives à l'implémentation .....	154
<i>Encapsulation d'une bibliothèque Java standard</i> .....	154
<i>Commandes du shell</i> .....	155
<i>Utilisation de Cron4J</i> .....	155
<i>Finalisation du projet</i> .....	155
Le code source .....	156
Planification de tâches .....	156
Encapsulation de SQLite .....	158
Le DAO SQLite .....	159
Descripteur de l'application .....	166
Ce qu'il faut en retenir .....	166

## ANNEXE

**Mettre en place un bundle OSGi avec PDE ..... 169****Index ..... 175**



# Avant-propos

---

Depuis environ deux ans, de nombreux produits voient leurs architectures complètement modifiées pour adopter une conception conforme à la norme OSGi. Que peut apporter cette norme au développeur et concepteur ? Que propose-t-elle ? Quel modèle et quels outils utiliser ?

## Pourquoi ce livre ?

Ce livre cherche à présenter en douceur la plate-forme OSGi, ainsi que l'environnement gravitant autour. Il comporte une section dédiée à l'outillage, ainsi qu'une section, plus rare en comparaison des ouvrages existants, consacrée aux aspects de conception en environnement OSGi.

Cet ouvrage ne tente pas d'entrer dans les arcanes d'un conteneur OSGi ni de détailler chacun des aspects de la norme ; il se veut une introduction cohérente à cette technologie.

Pour aller plus loin dans la technique, l'auteur ne peut que chaudement conseiller la lecture de l'indispensable *OSGi in Action* paru chez Manning et cosigné par Richard S. Hall, un des meilleurs spécialistes d'OSGi.

 Richard S. Hall, Karl Pauls et Stuart McCulloch, *OSGi in Action*, Manning, 2011

## À qui s'adresse ce livre ?

Cet ouvrage s'adresse aux étudiants, ingénieurs, architectes et chefs de projets expérimentés en Java et qui s'intéressent aux problématiques de modularité, ou qui doivent s'adapter à des contraintes liées au déploiement en environnement embarqué : développeurs JEE, développeurs RCP...

## Structure du livre

Après avoir esquissé le contexte architectural actuel, nous présenterons la notion de services propre au monde OSGi, puis nous aborderons la façon de travailler et de penser une application pour cette technologie. Enfin, nous illustrerons les concepts évoqués sur un cas pratique simple.

## Remerciements




L'auteur tient à remercier chaleureusement Peter Kriens, qui a toujours été à son écoute et a apporté des solutions sages et le plus souvent élégantes, tout en enrichissant ses réponses d'anecdotes croustillantes sur la face cachée d'une norme telle qu'OSGi. Il le remercie pour ses relectures précieuses, ainsi que Vincent Beretti et Christophe Peigné.

Il est naturel de remercier Karine Joly, Muriel Shan Sei Fan et Anne Bougnoux, le trio d'éditrices de choc ayant participé à la gestation puis à la finalisation de ce livre, ainsi que Sophie Hincelin, Anne-Lise Banéath et Gaël Thomas, qui ont œuvré avec minutie et efficacité lors des dernières étapes.

De plus, cet ouvrage n'aurait pas été possible sans la mission effectuée au sein de Telintrans.

Enfin, merci à ceux ayant subi les affres de la mauvaise humeur de l'auteur.

### BIBLIOGRAPHIE

-  R. S. Hall, K. Paul, S. McCulloch, *OSGi in Action*, Manning, 2011
-  E. Gamma et al., *Design Patterns: Elements of Reusable Software Components*, Addison Wesley, 1994
-  C. Beust, H. Suleiman, *Next Generation Java Testing*, Addison Wesley, 2007

# 1

## OSGi pour une conception modulaire

---

*OSGi (autrefois Open Services Gateway Initiative) est le nom d'une plate-forme orientée, à l'origine, vers l'embarqué, et dédiée à la gestion d'applications extrêmement configurables et dynamiques dans un environnement à une seule machine virtuelle Java. Ce premier chapitre a pour but de placer la plate-forme OSGi dans le contexte des applications modernes, de manière à justifier son utilisation dans des conditions a priori opposées : de l'application embarquée jusqu'au serveur d'applications. Il présente enfin les spécifications d'OSGi et les services qu'elle prévoit.*

### OSGi, pour quelles applications ?

Cette section examine quelques types d'applications contemporaines (automobile, environnement de développement, serveur d'applications) pour en extraire des besoins techniques communs. Cette collecte nous amènera naturellement vers une vue d'ensemble de la plate-forme OSGi.

## Cas d'une automobile

Objet de la vie de tous les jours, une voiture récente est un véritable écosystème d'applications informatiques, pour peu qu'elle dispose d'équipements du type :

- ABS (antiblocage de sécurité, système empêchant le blocage des roues en cas de freinage violent), dispositif nécessitant un logiciel à même de contrôler divers paramètres comme l'intensité du freinage, la vitesse, éventuellement le type de revêtement, etc. ;
- ESP (*Electronic Stability Program*, système de lutte contre le patinage, qui réduit le couple moteur dans certaines conditions de circulation où l'adhérence est problématique), dispositif qui requiert une application contrôlant le régime moteur en fonction d'un algorithme collectant ses entrées par le biais de capteurs ;
- lecteur CD ou MP3/DivX ;
- gestionnaire d'air conditionné ;
- etc.

Il existe ainsi des dizaines d'applications disséminées dans une automobile. Cet exemple nous donne l'occasion de formuler quelques remarques.

Tout d'abord, ces applications sont nombreuses et doivent cohabiter dans le même environnement, même si elles ne sont pas toutes du même niveau de criticité.

En outre, elles ont des cycles de vie fort distincts, puisque certaines sont résidentes (du démarrage jusqu'à l'arrêt du véhicule), alors que d'autres peuvent n'être effectives que pour quelques secondes.

Quant à l'environnement d'exécution, il est nécessairement cloisonné, de manière à ne pas voir une application critique tomber sous prétexte qu'une application d'importance moindre a provoqué des erreurs.

Enfin, ces applications forment un véritable microcosme et interagissent. En effet, plusieurs applications peuvent demander une modification du régime moteur, sans pour autant que le code correspondant soit dupliqué. Elles vont donc toutes s'adresser à une même application fournissant le service d'augmentation ou diminution du régime.

Pour citer Peter Kriens, le chantre d'OSGi, il est amusant de noter qu'un simple lecteur DivX représente le même nombre de lignes de code que l'informatique d'un programme spatial des années 1960, tandis que l'on constate que le volume du code à l'échelle planétaire double tous les 7 ans.

## Cas d'un environnement de développement (IDE)

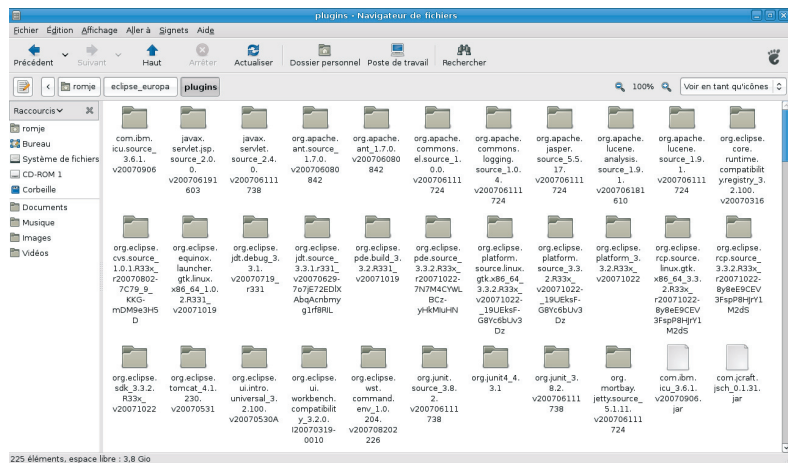
Quels sont les services attendus par un utilisateur d'IDE (*Integrated Development Environment*) ? Ils sont fort variés, même si en premier lieu, on s'attend à trouver des facilités pour :

- l'édition du code dans un langage donné ;
- le débogage de code ;
- le pilotage de l'exécution et du déploiement de nos applications.

Outre ces besoins courants, on peut rechercher une intégration avec des serveurs d'applications dans le monde Java, avec Apache dans le monde PHP. On peut avoir besoin d'outils de visualisation de traces, de consoles permettant un accès SSH (*Secure Shell*) vers un serveur, etc.

Comme il est impossible de dresser une liste exhaustive de ces besoins, l'architecture de tels produits doit donc prévoir un socle générique offrant diverses facilités transverses. En enrichissant les fonctionnalités de ce socle par l'ajout de greffons, on aboutit à l'idée de *plug-ins* présente dans Eclipse, ou encore à celle de *modules* dans le cadre de l'architecture NetBeans. Un IDE récent doit donc offrir un large éventail d'extensions et un cadre de développement de ces extensions de manière à couvrir de nombreux besoins distincts.

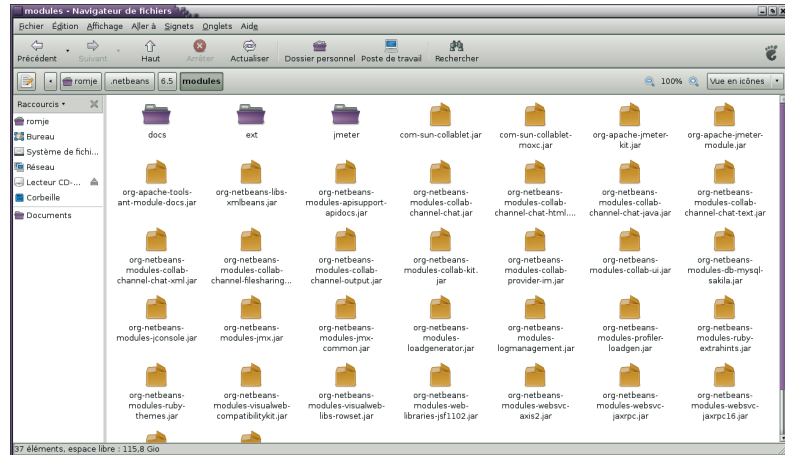
**Figure 1-1**  
Contenu du répertoire plugins  
pour Eclipse/Ganymede  
orienté J2EE



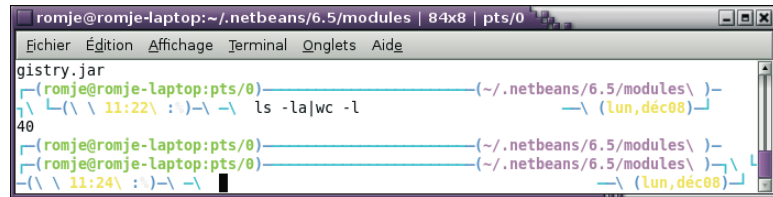
Les figures 1-1 à 1-4 montrent les contenus des répertoires d'extensions fournis avec les deux environnements majeurs de développement du monde Java : NetBeans et Eclipse. Ce dernier ne comporte pas moins de 583 plug-ins, alors que NetBeans couvre sensiblement le même spectre de fonctionnalités avec tout de même

40 modules (la granularité des modules NetBeans n'est pas exactement comparable à celle d'un plug-in ou bundle dans Eclipse) !

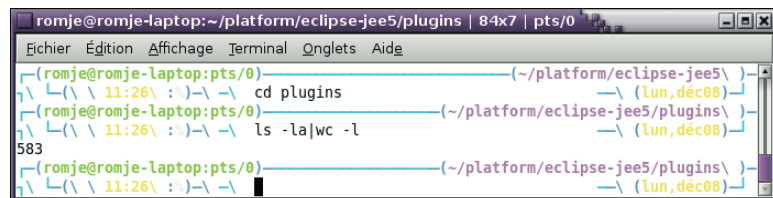
**Figure 1–2**  
Contenu du répertoire modules de NetBeans 6.5



**Figure 1–3**  
Nombre de modules dans un NetBeans 6.5/All



**Figure 1–4**  
Nombre de plug-ins dans Eclipse Ganymede JEE5 sous Linux



Évidemment, une telle profusion de besoins s'accompagne, parfois pour ces produits, de la nécessité de fonctionner sur des machines n'étant pas des bêtes de course. Il est donc impératif que l'architecture des IDE récents permette un chargement à la demande des extensions, afin de laisser disponible un volume maximal de mémoire pour un fonctionnement raisonnable du programme.

**DÉFINITION Lazy loading : le chargement à la demande**

Cette stratégie de chargement est opposée à une politique plus brutale, consistant à charger tout au démarrage. On retrouve ces noms de politiques dans des produits comme Hibernate. Il est logique de désactiver les ressources requises pour travailler en PHP quand on se prépare à travailler sur un serveur de sources Subversion, par exemple. Évidemment, cette stratégie impose de nombreux accès au disque ; tout avantage a sa contrepartie.

## Cas d'un serveur d'applications

Un tel serveur est un hôte pour vos applications ; il met à leur disposition diverses ressources :

- *pool* de connexions JDBC par le biais de DataSource JDBC ;
- annuaires LDAP par le biais de JNDI ;
- autres ERP par le biais de connecteurs JCA...

Le serveur fournit également des services techniques :

- sécurité avec JAAS ;
- persistance avec JPA ;
- transactions par JTA.

Les applications ainsi hébergées vont utiliser des bibliothèques Java et se doivent de disposer d'un taux de disponibilité maximal. Rationaliser leur utilisation, en évitant un chargement multiple de la même bibliothèque au gré des humeurs des développeurs créant les paquetages, rend plus robuste l'ensemble des applications.

Enfin, une autre fonctionnalité est souvent désirée concernant un serveur d'applications : le fait de pouvoir exposer diverses versions de la même application – chose impossible jusque récemment.

EN CLAIR	
JDBC	<i>Java DataBase Connectivity</i>
LDAP	<i>Lightweight Directory Access Protocol</i> : protocole de gestion d'annuaires de réseau
JNDI	<i>Java Naming and Directory Interface</i> : extension Java pour l'accès aux services d'annuaire
ERP	<i>Enterprise Resources Planning</i> : progiciel de gestion d'entreprise
JCA	<i>J2EE Connector Architecture</i> : interface d'accès aux applications Java
JAAS	<i>Java Authentification and Authorization Service</i>
JPA	<i>Java Persistence API</i>
JTA	<i>Java Transaction API</i>